Project Title:
Checkmate Revolution: A Four-Player Chess Variant with Rotating Boards

Submitted By:
Aazar Arnold (22K-4277), Bilal Ansari (22K-4365), Umer Nadeem (22K-4158)

Course:
AI

Instructor:
Ms. Alishba Subhani (Lab), Ms. Ramsha Jatt (Theory)

Submission Date:
16/04/2025

---

1. Project Overview

● Project Topic:
We propose a four-player chess variant called *Checkmate Revolution* featuring a rotating board and unconventional piece distribution. Each player controls a mini-army on one side of a square board, and every 4 turns, the board rotates 90°, shifting positional advantage and introducing dynamic strategy elements.

● Objective:
The main goal of this project is to develop a strategic AI using the **Minimax algorithm with Alpha-Beta pruning** to make optimal decisions in a four-player setting. The AI must adapt to constantly changing board states due to the rotation mechanic and track the evolving threat landscape from multiple opponents simultaneously.

---

2. Game Description

● Original Game Background:
Classic Chess is a two-player, zero-sum game where players control 16 pieces each on an 8x8 board. The objective is to checkmate the opponent's king through a combination of strategic movement, sacrifice, and positioning.

● Innovations Introduced:

- **Four Players:** Each player starts from one side of a square board and has 8 pieces: 1 King, 1 Queen, 1 Rook, 1 Bishop, 2 Knights, and 2 Pawns.

- **Rotating Board:** Every 4 turns (once all players have moved), the board rotates 90°, reorienting positions and opening or closing potential threats.
- **Team Mode or Free-for-All:** Optional toggle for team-based (2v2) or free-for-all play.
- **Piece Conversion:** If a player eliminates another's king, they get to "convert" one of the defeated player's pieces for their own use.

Impact on Gameplay Complexity and Strategy:
These changes introduce a high degree of spatial-temporal complexity. Players (and AI) must anticipate positional shifts caused by board rotation, navigate attacks from multiple fronts, and adjust strategies in real time. Tactical depth is increased exponentially due to multiparty dynamics and ever-changing spatial relationships.

---

3. AI Approach and Methodology

● AI Techniques to be Used:

- **Minimax Algorithm:** Extended to support four-player game trees, where each player aims to maximize their own utility rather than simply minimizing the opponent’s.
- **Alpha-Beta Pruning:** Adapted to reduce branching factor in a multiplayer scenario.
- **Optional Reinforcement Learning:** For future iterations, we may implement self-play RL to allow the AI to discover non-obvious strategies.

● Heuristic Design:

- Piece value weighting (King: infinite, Queen: 9, Rook: 5, etc.)
- Positional value based on board quadrant control
- Threat detection (proximity of enemy pieces post-rotation)
- Conversion opportunity scoring

● Complexity Analysis:
The multiplayer game tree exponentially increases with each added player. Estimating a standard branching factor of 30 moves per player, the game tree becomes:

- **Time Complexity (Minimax):** O(b^d), where b = ~30, d = ~3–4 plies for manageable performance.
- **Challenges:** Multiplayer strategy divergence, board rotation re-mapping, and predicting interactions between other AIs or human players.

---

4. Game Rules and Mechanics

● Modified Rules:

- 4 players instead of 2
- Each player controls 8 pieces
- Board rotates 90° clockwise every full round
- Piece conversion upon enemy king capture

● **Winning Conditions:**

- **Free-for-All:** Last player with a king standing wins
- **Team Mode:** Eliminate both opponents' kings
- Optional sudden-death mode if game exceeds 50 turns

● **Turn Sequence:**

- Turn order: Player 1 → Player 2 → Player 3 → Player 4
- After all four have moved, the board rotates
- Game continues in clockwise sequence

---

5. Implementation Plan

● **Programming Language:**
Python

● **Libraries and Tools:**

- **Pygame** for GUI
- **NumPy** for board state management
- **Matplotlib** for visualizing AI decision trees (optional)
- **Custom-built logic engine** for AI and rotation mechanics

● **Milestones and Timeline:**

- **Week 1-2:** Finalize game rules, board design, and layout
- **Week 3-4:** Implement basic movement and piece logic
- **Week 5-6:** Develop and test AI using Minimax with heuristics
- **Week 7:** Integrate AI with rotation mechanic and UI
- **Week 8:** Final testing, optimization, and report

---

6. References

- Norvig, P., & Russell, S. (2021). *Artificial Intelligence: A Modern Approach*.
- Silver, D. et al. (2018). *A General Reinforcement Learning Algorithm that Masters Chess*.

- Pygame Documentation
- Python Chess Library
- Red Blob Games – Game AI visualizations
- YouTube: Sebastian Lague's Coding Adventure – Chess AI series